

SUBSTITUTE FORM PTO-1390

U.S. DEPARTMENT OF COMMERCE PATENT AND TRADEMARK OFFICE

**TRANSMITTAL LETTER TO THE UNITED STATES  
DESIGNATED/ELECTED OFFICE (DO/EO/US)  
CONCERNING A FILING UNDER 35 U.S.C. 371**

ATTORNEY'S DOCKET NUMBER  
10559-311US1

U.S. APPLICATION NO. (If Known, see 37 CFR 1.5)

**10/070008**

INTERNATIONAL APPLICATION NO.  
PCT/US00/24006

INTERNATIONAL FILING DATE  
31 August 2000

PRIORITY DATE CLAIMED  
1 September 1999

TITLE OF INVENTION  
BRANCH INSTRUCTION FOR PROCESSOR

APPLICANT(S) FOR DO/EO/US  
Gilbert Wolrich, Matthew J. Adiletta, William R. Wheeler, Debra Bernstein and Donald F. Hooper

Applicant herewith submits to the United States Designated/Elected Office (DO/EO/US) the following items and other information:

1. ☒ This is a **FIRST** submission of items concerning a filing under 35 U.S.C. 371.
2. ☐ This is a **SECOND** or **SUBSEQUENT** submission of items concerning a filing under 35 U.S.C. 371.
3. ☒ This is an express request to promptly begin national examination procedures (35 U.S.C. 371(f)).
4. ☒ The US has been elected by the expiration of 19 months from the priority date (PCT Article 31).
5. ☒ A copy of the International Application as filed (35 U.S.C. 371(c)(2))
  - a. ☒ is attached hereto (required only if not communicated by the International Bureau).
  - b. ☐ has been communicated by the International Bureau.
  - c. ☐ is not required, as the application was filed in the United States Receiving Office (RO/US).
6. ☐ An English language translation of the International Application as filed (35 U.S.C. 371(c)(2)).
7. ☒ Amendments to the claims of the International Application under PCT Article 19 (35 U.S.C. 371(c)(3))
  - a. ☐ are attached hereto (required only if not communicated by the International Bureau).
  - b. ☐ have been communicated by the International Bureau.
  - c. ☐ have not been made; however, the time limit for making such amendments has NOT expired.
  - d. ☒ have not been made and will not be made.
8. ☐ An English language translation of amendments to the claims under PCT Article 19 (35 U.S.C. 371(c)(3)).
9. ☐ An oath or declaration of the inventor(s) (35 U.S.C. 371(c)(4)).
10. ☐ An English language translation of the annexes to the International Preliminary Examination Report under PCT Article 36 (35 U.S.C. 371(c)(5)).

**Items 11 to 16 below concern other documents or information included:**

11. ☒ An Information Disclosure Statement under 37 CFR 1.97 and 1.98.
12. ☐ An assignment document for recording. A separate cover sheet in compliance with 37 CFR 3.28 and 3.31 is included.
13. ☐ A FIRST preliminary amendment.  
☐ A SECOND or SUBSEQUENT preliminary amendment.
14. ☐ A substitute specification.
15. ☐ A change of power of attorney and/or address letter.
16. ☐ Other items or information:

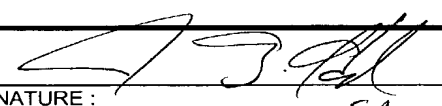
☐  
☐  
☐  
☐  
☐

**CERTIFICATE OF MAILING BY EXPRESS MAIL**

Express Mail Label No. **EV044492476US**

I hereby certify under 37 CFR §1.10 that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, DC 20231

**2-28-00**  **Gabriel Lewis**  
Date of Deposit Signature Typed Name of Person Signing

U.S. APPLICATION NO. (IF KNOWN) <b>10/ 0700008</b>		INTERNATIONAL APPLICATION NO. PCT/US00/24006		ATTORNEY'S DOCKET NUMBER 10559-311US1																					
17. <input checked="" type="checkbox"/> The following fees are submitted:  <b>Basic National Fee ( 37 CFR 1.492(a)(1)-( 5) ):</b>  Neither international preliminary examination fee (37 CFR 1.482) nor international search fee (37 CFR 1.445(a)(2)) paid to USPTO and International Search Report not prepared by the EPO or JPO..... <b>\$1040</b>  International preliminary examination fee (37 CFR 1.482) not paid to USPTO but International Search Report prepared by the EPO or JPO ..... <b>\$890</b>  International preliminary examination fee (37 CFR 1.482) not paid to USPTO but international search fee (37 CFR 1.445(a)(2)) paid to USPTO..... <b>\$740</b>  International preliminary examination fee paid to USPTO (37 CFR 1.482) but all claims did not satisfy provisions of PCT Article 33(1)-(4)..... <b>\$710</b>  International preliminary examination fee paid to USPTO (37 CFR 1.482) and all claims satisfied provisions of PCT Article 33(1)-(4) ..... <b>\$100</b>  <div style="text-align: right;"><b>ENTER APPROPRIATE BASIC FEE AMOUNT =</b></div>				<b>CALCULATIONS PTO USE ONLY</b>																					
Surcharge of <b>\$130</b> for furnishing the oath or declaration later than <input type="checkbox"/> 20 <input checked="" type="checkbox"/> 30 months from the earliest claimed priority date (37 CFR 1.492(e)).				\$100.00																					
<table border="1" style="width:100%; border-collapse: collapse;"> <thead> <tr> <th style="width:20%;">Claims</th> <th style="width:20%;">Number Filed</th> <th style="width:20%;">Number Extra</th> <th style="width:20%;">Rate</th> </tr> </thead> <tbody> <tr> <td>Total Claims</td> <td>21 - 20 =</td> <td>1</td> <td>x <b>\$18</b></td> </tr> <tr> <td>Independent Claims</td> <td>3 - 3 =</td> <td>0</td> <td>x <b>\$84</b></td> </tr> <tr> <td colspan="3">MULTIPLE DEPENDENT CLAIMS(S) (if applicable)</td> <td>+ <b>\$280</b></td> </tr> <tr> <td colspan="3" style="text-align: right;"><b>TOTAL OF ABOVE CALCULATIONS =</b></td> <td></td> </tr> </tbody> </table>				Claims	Number Filed	Number Extra	Rate	Total Claims	21 - 20 =	1	x <b>\$18</b>	Independent Claims	3 - 3 =	0	x <b>\$84</b>	MULTIPLE DEPENDENT CLAIMS(S) (if applicable)			+ <b>\$280</b>	<b>TOTAL OF ABOVE CALCULATIONS =</b>				\$130.00	
Claims	Number Filed	Number Extra	Rate																						
Total Claims	21 - 20 =	1	x <b>\$18</b>																						
Independent Claims	3 - 3 =	0	x <b>\$84</b>																						
MULTIPLE DEPENDENT CLAIMS(S) (if applicable)			+ <b>\$280</b>																						
<b>TOTAL OF ABOVE CALCULATIONS =</b>																									
				\$18.00																					
				\$0.00																					
				\$0.00																					
				\$248.00																					
<input type="checkbox"/> Applicant claims small entity status. See 37 CFR 1.27. The fees indicated above are reduced by 1/2.				\$0.00																					
<b>SUBTOTAL =</b>				\$248.00																					
Processing fee of <b>\$130</b> for furnishing the English Translation later than <input type="checkbox"/> 20 <input type="checkbox"/> 30 months from the earliest claimed priority date (37 CFR 1.492(f))				\$0.00																					
<b>TOTAL NATIONAL FEE =</b>				\$248.00																					
Fee for recording the enclosed assignment (37 CFR 1.21(h)). The assignment must be accompanied by an appropriate cover sheet (37 CFR 3.28, 3.31). <b>\$40.00</b> per property +				\$0.00																					
<b>TOTAL FEES ENCLOSED =</b>				\$248.00																					
				<b>Amount to be refunded:</b>	\$																				
				<b>Charged:</b>	\$																				
a. <input checked="" type="checkbox"/> A check in the amount of \$248.00 to cover the above fees is enclosed. b. <input type="checkbox"/> Please charge my Deposit Account No. 06-1050 in the amount of \$0.00 to cover the above fees. A duplicate copy of this sheet is enclosed. c. <input checked="" type="checkbox"/> The Commissioner is hereby authorized to charge any additional fees which may be required, or credit any overpayment to Deposit Account No. 06-1050. A duplicate copy of this sheet is enclosed.																									
<b>NOTE: Where an appropriate time limit under 37 CFR 1.494 or 1.495 has not been met, a petition to revive (37 CFR 1.137(a) or (b) must be filed and granted to restore the application to pending status.</b>																									
SEND ALL CORRESPONDENCE TO:																									
Scott C. Harris FISH & RICHARDSON P.C. 4350 La Jolla Village Drive, Suite 500 San Diego, California 92122 (858) 678-5070 phone (858) 678-5099 facsimile			<div style="text-align: center;">             SIGNATURE: <b>JAMES T. HAGLER</b>            NAME: <b>REG. NO 40,631</b>            REGISTRATION NUMBER: <b>32,030</b> </div>																						

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization  
International Bureau(43) International Publication Date  
8 March 2001 (08.03.2001)

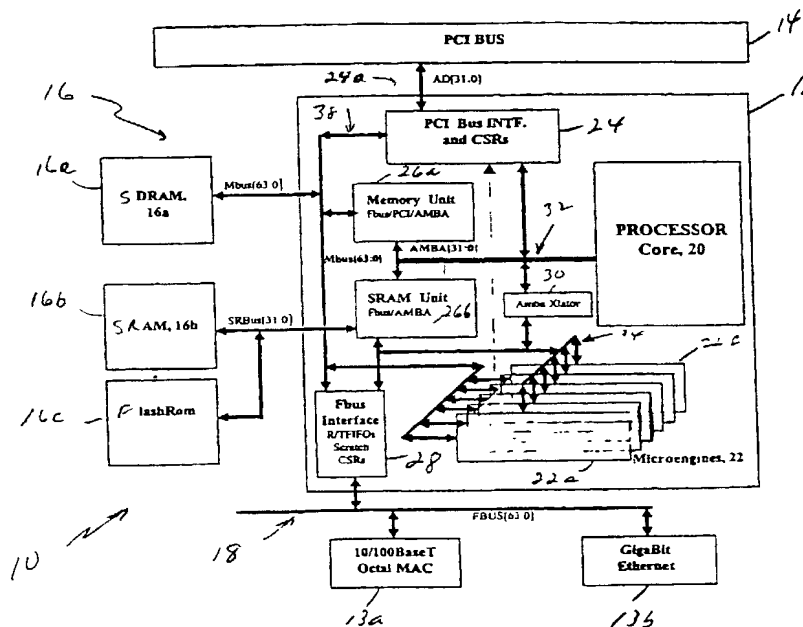
PCT

(10) International Publication Number  
**WO 01/16713 A1**(51) International Patent Classification<sup>7</sup>: **G06F 9/30**(21) International Application Number: **PCT/US00/24006**(22) International Filing Date: **31 August 2000 (31.08.2000)**(25) Filing Language: **English**(26) Publication Language: **English**(30) Priority Data:  
60/151,961 1 September 1999 (01.09.1999) **US**

(63) Related by continuation (CON) or continuation-in-part (CIP) to earlier application:

**US** 60/151,961 (CIP)  
Filed on 1 September 1999 (01.09.1999)(71) Applicant (for all designated States except US): **INTEL CORPORATION [US/US]; 2200 Mission College Boulevard, Santa Clara, CA 95052 (US).**

(72) Inventors; and

(75) Inventors/Applicants (for US only): **WOLRICH, Gilbert [US/US]; 4 Cider Mill Road, Framingham, MA**01701 (US). **ADILETTA, Matthew, J. [US/US]**, 20 Monticello Drive, Worcester, MA 01603 (US). **WHEELER, William [US/US]**, 745 School Street, Webster, MA 01570 (US). **BERNSTEIN, Debra [US/US]**, 38 Helen Street, Waltham, MA 02452 (US). **HOOPEER, Donald [US/US]**, 19 Main Circle, Shrewsbury, MA 01545 (US).(74) Agents: **MALONEY, Denis, G.; Fish & Richardson P.C., 225 Franklin Street, Boston, MA 02110-2804 et al. (US).**(81) Designated States (national): **AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.**(84) Designated States (regional): **ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).***[Continued on next page]*(54) Title: **BRANCH INSTRUCTION FOR PROCESSOR**

(57) Abstract: A processor such as a multithreaded hardware based processor (12) is described. The processor (12) includes a computer instruction (20) that is a branch instruction that causes a branch in execution of an instruction stream based on any specified value being true or false. The instruction also includes a token that specifies the number of instruction of instructions following the branch instruction to execute before performing the branch operation.

## BRANCH INSTRUCTION FOR PROCESSOR

## BACKGROUND

This invention relates to branch instructions.

5 Parallel processing is an efficient form of information processing of concurrent events in a computing process. Parallel processing demands concurrent execution of many programs in a computer. Sequential processing or serial processing has all tasks performed sequentially at a single station whereas, pipelined processing has tasks performed at specialized stations. Computer code whether executed in parallel  
10 processing, pipelined or sequential processing machines involves branches in which an instruction stream may execute in a sequence and branch from the sequence to a different sequence of instructions.

## BRIEF DESCRIPTION OF THE DRAWINGS

15 FIG. 1 is a block diagram of a communication system employing a processor.

FIG. 2 is a detailed block diagram of the.

FIG. 3 is a block diagram of a microengine used in the processor of FIGS.  
1 and 2.

20 FIG. 4 is a block diagram of a pipeline in the microengine of FIG. 3.

FIGS. 5A-5C show exemplary formats for branch instructions.

FIG. 6 is a block diagram of general purpose registers.

## DESCRIPTION

25 Referring to FIG. 1, a communication system 10 includes a processor 12. In one embodiment, the processor is a hardware-based multithreaded processor 12. The processor 12 is coupled to a bus such as a PCI bus 14, a memory system 16 and a second bus 18. The system 10 is especially useful for tasks that can be broken into parallel sub-tasks or functions. Specifically hardware-based multithreaded processor 12 is useful for  
30 tasks that are bandwidth oriented rather than latency oriented. The hardware-based multithreaded processor 12 has multiple microengines 22 each with multiple hardware controlled threads that can be simultaneously active and independently work on a task.

The hardware-based multithreaded processor 12 also includes a central controller 20 that assists in loading microcode control for other resources of the hardware-based multithreaded processor 12 and performs other general purpose computer type functions such as handling protocols, exceptions, extra support for packet processing where the microengines pass the packets off for more detailed processing such as in boundary conditions. In one embodiment, the processor 20 is a Strong Arm<sup>®</sup> (Arm is a trademark of ARM Limited, United Kingdom) based architecture. The general purpose microprocessor 20 has an operating system. Through the operating system the processor 20 can call functions to operate on microengines 22a-22f. The processor 20 can use any supported operating system preferably a real time operating system. For the core processor implemented as a Strong Arm architecture, operating systems such as, MicrosoftNT<sup>®</sup> real-time, VXWorks and  $\square$ CUS, a freeware operating system available over the Internet, can be used.

The hardware-based multithreaded processor 12 also includes a plurality of function microengines 22a-22f. Functional microengines (microengines) 22a-22f each maintain a plurality of program counters in hardware and states associated with the program counters. Effectively, a corresponding plurality of sets of threads can be simultaneously active on each of the microengines 22a-22f while only one is actually operating at any one time.

Microengines 22a-22f each have capabilities for processing four hardware threads. The microengines 22a-22f operate with shared resources including memory system 16 and bus interfaces 24 and 28. The memory system 16 includes a Synchronous Dynamic Random Access Memory (SDRAM) controller 26a and a Static Random Access Memory (SRAM) controller 26b. SDRAM memory 16a and SDRAM controller 26a are typically used for processing large volumes of data, e.g., processing of network payloads from network packets. The SRAM controller 26b and SRAM memory 16b are used in, e.g., networking packet processing, postscript processor, or as a processor for a storage subsystem, i.e., RAID disk storage, or for low latency, fast access tasks, e.g., accessing look-up tables, memory for the core processor 20, and so forth.

The processor 12 includes a bus interface 28 that couples the processor to the second bus 18. Bus interface 28 in one embodiment couples the processor 12 to the so-called FBUS 18 (FIFO bus). The processor 12 includes a second interface e.g., a PCI bus interface 24 that couples other system components that reside on the PCI 14 bus to

the processor 12. The PCI bus interface 24, provides a high speed data path 24a to the SDRAM memory 16a. Through that path data can be moved quickly from the SDRAM 16a through the PCI bus 14, via direct memory access (DMA) transfers.

Each of the functional units are coupled to one or more internal buses.

5 The internal buses are dual, 32 bit buses (i.e., one bus for read and one for write). The hardware-based multithreaded processor 12 also is constructed such that the sum of the bandwidths of the internal buses in the processor 12 exceed the bandwidth of external buses coupled to the processor 12. The processor 12 includes an internal core processor bus 32, e.g., an ASB bus (Advanced System Bus) that couples the processor core 20 to  
10 the memory controller 26a, 26c and to an ASB translator 30 described below. The ASB bus is a subset of the so called AMBA bus that is used with the Strong Arm processor core. The processor 12 also includes a private bus 34 that couples the microengine units to SRAM controller 26b, ASB translator 30 and FBUS interface 28. A memory bus 38 couples the memory controller 26a, 26b to the bus interfaces 24 and 28 and memory  
15 system 16 including flashrom 16c used for boot operations and so forth.

Referring to FIG. 2, each of the microengines 22a-22f includes an arbiter that examines flags to determine the available threads to be operated upon. Any thread from any of the microengines 22a-22f can access the SDRAM controller 26a, SDRAM controller 26b or FBUS interface 28. The memory controllers 26a and 26b each include a  
20 plurality of queues to store outstanding memory reference requests. The FBUS interface 28 supports Transmit and Receive flags for each port that a MAC device supports, along with an Interrupt flag indicating when service is warranted. The FBUS interface 28 also includes a controller 28a that performs header processing of incoming packets from the FBUS 18. The controller 28a extracts the packet headers and performs a  
25 microprogrammable source/destination/protocol hashed lookup (used for address smoothing) in SRAM.

The core processor 20 accesses the shared resources. The core processor 20 has a direct communication to the SDRAM controller 26a to the bus interface 24 and to SRAM controller 26b via bus 32. However, to access the microengines 22a-22f and  
30 transfer registers located at any of the microengines 22a-22f, the core processor 20 access the microengines 22a-22f via the ASB Translator 30 over bus 34. The ASB translator 30 can physically reside in the FBUS interface 28, but logically is distinct. The ASB Translator 30 performs an address translation between FBUS microengine transfer

100743003 10740302

register locations and core processor addresses (i.e., ASB bus) so that the core processor 20 can access registers belonging to the microengines 22a-22c.

Although microengines 22 can use the register set to exchange data as described below, a scratchpad memory 27 is also provided to permit microengines to write data out to the memory for other microengines to read. The scratchpad 27 is coupled to bus 34.

The processor core 20 includes a RISC core 50 implemented in a five stage pipeline performing a single cycle shift of one operand or two operands in a single cycle, provides multiplication support and 32 bit barrel shift support. This RISC core 50 is a standard Strong Arm® architecture but it is implemented with a five stage pipeline for performance reasons. The processor core 20 also includes a 16 kilobyte instruction cache 52, an 8 kilobyte data cache 54 and a prefetch stream buffer 56. The core processor 20 performs arithmetic operations in parallel with memory writes and instruction fetches. The core processor 20 interfaces with other functional units via the ARM defined ASB bus. The ASB bus is a 32-bit bi-directional bus 32.

Referring to FIG. 3, an exemplary microengine 22f includes a control store 70 that includes a RAM which stores a microprogram. The microprogram is loadable by the core processor 20. The microengine 22f also includes controller logic 72. The controller logic includes an instruction decoder 73 and program counter (PC) units 72a-72d. The four micro program counters 72a-72d are maintained in hardware. The microengine 22f also includes context event switching logic 74. Context event logic 74 receives messages (e.g., SEQ\_#\_EVENT\_RESPONSE; FBI\_EVENT\_RESPONSE; SRAM\_EVENT\_RESPONSE; SDRAM\_EVENT\_RESPONSE; and ASB\_EVENT\_RESPONSE) from each one of the shared resources, e.g., SRAM 26a, SDRAM 26b, or processor core 20, control and status registers, and so forth. These messages provide information on whether a requested function has completed. Based on whether or not a function requested by a thread has completed and signaled completion, the thread needs to wait for that completion signal, and if the thread is enabled to operate, then the thread is placed on an available thread list (not shown). The microengine 22f can have a maximum of e.g., 4 threads available.

In addition to event signals that are local to an executing thread, the microengines 22 employ signaling states that are global. With signaling states, an executing thread can broadcast a signal state to all microengines 22. Receive Request

Available signal, Any and all threads in the microengines can branch on these signaling states. These signaling states can be used to determine availability of a resource or whether a resource is due for servicing.

The context event logic 74 has arbitration for the four (4) threads. In one  
 5 embodiment, the arbitration is a round robin mechanism. Other techniques could be used including priority queuing or weighted fair queuing. The microengine 22f also includes an execution box (EBOX) data path 76 that includes an arithmetic logic unit 76a and general purpose register set 76b. The arithmetic logic unit 76a performs arithmetic and logical functions as well as shift functions. The arithmetic logic unit includes condition  
 10 code bits that are used by instructions described below. The registers set 76b has a relatively large number of general purpose registers that are windowed as will be described so that they are relatively and absolutely addressable. The microengine 22f also includes a write transfer register stack 78 and a read transfer stack 80. These registers are also windowed so that they are relatively and absolutely addressable. Write transfer  
 15 register stack 78 is where write data to a resource is located. Similarly, read register stack 80 is for return data from a shared resource. Subsequent to or concurrent with data arrival, an event signal from the respective shared resource e.g., the SRAM controller 26a, SDRAM controller 26b or core processor 20 will be provided to context event arbiter 74 which will then alert the thread that the data is available or has been sent. Both  
 20 transfer register banks 78 and 80 are connected to the execution box (EBOX) 76 through a data path.

Referring to FIG. 4, the microengine datapath maintains a 5-stage micro-pipeline 82. This pipeline includes lookup of microinstruction words 82a, formation of the register file addresses 82b, read of operands from register file 82c, ALU, shift or  
 25 compare operations 82d, and write-back of results to registers 82e. By providing a write-back data bypass into the ALU/shifter units, and by assuming the registers are implemented as a register file (rather than a RAM), the microengine can perform a simultaneous register file read and write, which completely hides the write operation.

The instruction set supported in the microengines 22a-22f support  
 30 conditional branches. The worst case conditional branch latency (not including jumps) occurs when the branch decision is a result of condition codes being set by the previous microcontrol instruction. The latency is shown below in Table 1:



TABLE 1

		1	2	3	4	5	6	7	8
		-----+	-----+	-----+	-----+	-----+	-----+	-----+	-----+
5	microstore lookup	n1	cb	n2	XX	b1	b2	b3	b4
	reg addr gen		n1	cb	XX	XX	b1	b2	b3
	reg file lookup			n1	cb	XX	XX	b1	b2
	ALU/shifter/cc				n1	cb	XX	XX	b1
	write back			m2		n1	cb	XX	XX

10

where nx is pre-branch microword (n1 sets cc's)

cb is conditional branch

bx is post-branch microword

XX is aborted microword

15

As shown in Table 1, it is not until cycle 4 that the condition codes of n1 are set, and the branch decision can be made (which in this case causes the branch path to be looked up in cycle 5). The microengine 22f incurs a 2-cycle branch latency penalty because it must abort operations n2 and n3 (the 2 microwords directly after the branch) in the pipe, before the branch path begins to fill the pipe with operation b1. If the branch is not taken, no microwords are aborted and execution continues normally. The microengines have several mechanisms to reduce or eliminate the effective branch latency.

The microengines support selectable deferred branches. Selectable deferring branches are when a microengine allows 1 or 2 micro instructions after the branch to execute before the branch takes effect (i.e. the effect of the branch is "deferred" in time). Thus, if useful work can be found to fill the wasted cycles after the branch microword, then the branch latency can be hidden. A 1-cycle deferred branch is shown below in Table 2 where n2 is allowed to execute after cb, but before b1:

30

TABLE 2

		1	2	3	4	5	6	7	8	
		-----+	-----+	-----+	-----+	-----+	-----+	-----+	-----+	
5	microstore lookup	n1	cb	n2	XX	b1	b2	b3	b4	
	reg addr gen		n1	cb	n2	XX	b1	b2	b3	
	reg file lookup			n1	cb	n2	XX	b1	b2	
	ALU/shifter/cc				n1	cb	n2	XX	b1	
	write back					n1	cb	n2	XX	

10

A 2-cycle deferred branch is shown in TABLE 3 where n2 and n3 are both allowed to complete before the branch to b1 occurs. Note that a 2-cycle branch deferment is only allowed when the condition codes are set on the microword preceding the branch.

15

TABLE 3

		1	2	3	4	5	6	7	8	9	
		-----+	-----+	-----+	-----+	-----+	-----+	-----+	-----+	-----+	
	microstore lookup	n1	cb	n2	n3	b1	b2	b3	b4	b5	
20	reg addr gen		n1	cb	n2	n3	b1	b2	b3	b4	
	reg file lkup			n1	cb	n2	n3	b1	b2	b3	
	ALU/shftr/cc				n1	cb	n2	n3	b1	b2	
	write back					n1	cb	n2	n3	b1	

25

The microengines also support condition code evaluation. If the condition codes upon which a branch decision are made are set 2 or more microwords before the branch, then 1 cycle of branch latency can be eliminated because the branch decision can be made 1 cycle earlier as in Table 4.

TABLE 4

	1	2	3	4	5	6	7	8
5	-----	+	-----	+	-----	+	-----	+
	microstore lookup	n1	n2	cb	XX	b1	b2	b3   b4
	reg addr gen		n1	n2	cb	XX	b1	b2   b3
	reg file lookup			n1	n2	cb	XX	b1   b2
	ALU/shifter/cc				n1	n2	cb	XX   b1
10	write back					n1	n2	cb   XX

In this example, n1 sets the condition codes and n2 does not set the conditions codes. Therefore, the branch decision can be made at cycle 4 (rather than 5), to eliminate 1 cycle of branch latency. In the example in Table 5 the 1-cycle branch deferment and early setting of condition codes are combined to completely hide the branch latency. That is, the condition codes (cc's) are set 2 cycles before a 1-cycle deferred branch.

TABLE 5

	1	2	3	4	5	6	7	8
20	-----	+	-----	+	-----	+	-----	+
	microstore lookup	n1	n2	cb	n3	b1	b2	b3   b4
	reg addr gen		n1	n2	cb	n3	b1	b2   b3
	reg file lookup			n1	n2	cb	n3	b1   b2
	ALU/shifter/cc				n1	n2	cb	n3   b1
25	write back					n1	n2	cb   n3

In the case where the condition codes cannot be set early (i.e. they are set in the microword preceding the branch), the microengine supports branch guessing which attempts to reduce the 1 cycle of exposed branch latency that remains. By "guessing" the branch path or the sequential path, the microsequencer pre-fetches the guessed path 1 cycle before it definitely knows what path to execute. If it guessed correctly, 1 cycle of branch latency is eliminated as shown in Table 6.

TABLE 6

guess branch taken /branch is taken		1   2   3   4   5   6   7   8							
5		-----+-----+-----+-----+-----+-----+-----+-----+							
	microstore lookup	n1	cb	n1	b1	b2	b3	b4	b5
	reg addr gen		n1	cb	XX	b1	b2	b3	b4
	reg file lookup			n1	cb	XX	b1	b2	b3
	ALU/shifter/cc				n1	cb	XX	b1	b2
10	write back					n1	cb	XX	b1

If the microcode guessed a branch taken incorrectly, the microengine still only wastes 1 cycle as in TABLE 7

TABLE 7

guess branch taken /branch is NOT taken		1   2   3   4   5   6   7   8							
20		-----+-----+-----+-----+-----+-----+-----+-----+							
	microstore lookup	n1	cb	n1	XX	n2	n3	n4	n5
	reg addr gen		n1	cb	n1	XX	n2	n3	n4
	reg file lookup			n1	cb	n1	XX	n2	n3
	ALU/shifter/cc				n1	cb	n1	XX	n2
25	write back					n1	cb	n1	XX

However, the latency penalty is distributed differently when microcode guesses a branch is not taken. For guess branch NOT taken / branch is NOT taken there are no wasted cycles as in Table 8.

Table 8

		1	2	3	4	5	6	7	8
		-----+	-----+	-----+	-----+	-----+	-----+	-----+	-----+
5	microstore lookup	n1	cb	n1	n2	n3	n4	n5	n6
	reg addr gen		n1	cb	n1	n2	n3	n4	n5
	reg file lookup			n1	cb	n1	n2	n1	b4
	ALU/shifter/cc				n1	cb	n1	n2	n3
	write back					n1	cb	n1	n2

10

However for guess branch NOT taken /branch is taken there are 2 wasted cycles as in Table 9.

Table 9

		1	2	3	4	5	6	7	8
		-----+	-----+	-----+	-----+	-----+	-----+	-----+	-----+
15	microstore lookup	n1	cb	n1	XX	b1	b2	b3	b4
	reg addr gen		n1	cb	XX	XX	b1	b2	b3
20	reg file lookup			n1	cb	XX	XX	b1	b2
	ALU/shifter/cc				n1	cb	XX	XX	b1
	write back					n1	cb	XX	XX

The microengine can combine branch guessing with 1-cycle branch deferment to improve the result further. For guess branch taken with 1-cycle deferred branch/branch is taken is in Table 10.

Table 10

		1	2	3	4	5	6	7	8
5		-----+-----+-----+-----+-----+-----+-----+-----+							
	microstore lookup	n1	cb	n2	b1	b2	b3	b4	b5
	reg addr gen		n1	cb	n2	b1	b2	b3	b4
	reg file lookup			n1	cb	n2	b1	b2	b3
	ALU/shifter/cc				n1	cb	n2	b1	b2
10	write back					n1	cb	n2	b1

In the case above, the 2 cycles of branch latency are hidden by the execution of n2, and by correctly guessing the branch direction.

If microcode guesses incorrectly, 1 cycle of branch latency remains exposed as in Table 11 (guess branch taken with 1-cycle deferred branch/branch is NOT taken).

Table 11

		1	2	3	4	5	6	7	8	9
20		-----+-----+-----+-----+-----+-----+-----+-----+								
	microstore lookup	n1	cb	n2	XX	n3	n4	n5	n6	n7
	reg addr gen		n1	cb	n2	XX	n3	n4	n5	n6
	reg file lkup			n1	cb	n2	XX	n3	n4	n5
25	ALU/shftr/cc				n1	cb	n2	XX	n3	n4
	write back					n1	cb	n2	XX	n3

If microcode correctly guesses a branch NOT taken, then the pipeline flows sequentially in the normal unperturbed case. If microcode incorrectly guesses branch NOT taken, the microengine again exposes 1 cycle of unproductive execution as shown in Table 12.

Table 12

guess branch NOT taken/branch is taken

5		1   2   3   4   5   6   7   8   9
		-----+-----+-----+-----+-----+-----+-----+-----+-----+
	microstore lookup	n1   cb   n2   XX   b1   b2   b3   b4   b5
	reg addr gen	n1   cb   n2   XX   b1   b2   b3   b4
	reg file lkup	n1   cb   n2   XX   b1   b2   b3
10	ALU/shftr/cc	n1   cb   n2   XX   b1   b2
	write back	n1   cb   n2   XX   b1
	where nx is pre-branch microword (n1 sets cc's)	
	cb is conditional branch	
	bx is post-branch microword	
15	XX is aborted microword	

In the case of a jump instruction, 3 extra cycles of latency are incurred because the branch address is not known until the end of the cycle in which the jump is in the ALU stage (Table 13).

Table 13

20		1   2   3   4   5   6   7   8   9
		-----+-----+-----+-----+-----+-----+-----+-----+-----+
	microstore lookup	n1   jp   XX   XX   XX   j1   j2   j3   j4
25	reg addr gen	n1   jp   XX   XX   XX   j1   j2   j3
	reg file lkup	n1   jp   XX   XX   XX   j1   j2
	ALU/shftr/cc	n1   jp   XX   XX   XX   j1
	write back	n1   jp   XX   XX   XX

30 Conditional branches that operate on ALU condition codes which are set on the microword before the branch can select 0, 1 or 2 or 3 cycle branch deferment modes. Condition codes set 2 or more microwords before the conditional branch that operates on them can select 0 or 1-cycle branch deferment modes. All other branches

(including context rearrbitrations) can select either 0 or 1-cycle branch deferment modes. The architecture could be designed to make a context arbitration microword within a branch deferment window of a preceding branch, jump or context arbitration microword, an illegal option. That is, in some embodiments, a context switch would not be allowed to occur during a branch transition in the pipeline because it could unduly complicate saving of the old context PC. The architecture could also be designed to make branching within the branch deferment window of a preceding branch, jump or context arbitration microword illegal to avoid complicated and possible unpredictable branch behaviors.

Referring to FIG. 5A, general formats for branch instructions are shown.

#### 10 Branch Instructions

BR BR<0, BR=0 BR<=0. BR!=0 BR=COUT, BR>0 BR!=COUT, BR>=0

One set of branch instructions are branch unconditionally BR or branch to an instruction at a specified label based on an ALU condition code. Exemplary ALU condition codes are Sign, Zero, and Carryout (cout).

#### 15 Formats: br[label#], optional\_token

br=0[label#], optional\_token

br!=0[label#], optional\_token

br>0[label#], optional\_token

br>=0[label#], optional\_token

#### 20 br<0[label#], optional\_token

br<=0[label#], optional\_token

br=cout[label#], optional\_token

br!=cout[label#], optional\_token

The parameter label# is a symbolic label corresponding to the address of an instruction. This class of branch instructions can have the following optional\_tokens; defer one instruction, defer two instructions, and defer three instructions. These optional tokens cause the processor to execute one, two, or three instructions following this instruction before performing the branch operation. The ALU operation that sets the condition codes may occur several instructions before the branch instruction. The defer two and defer three are not used with branch guess as described below

A fourth optional token is the guess\_branch token which causes the processor to prefetch the instruction for the "branch taken" condition rather than the next



sequential instruction. The ALU operation that sets the condition codes occurs immediately before the branch instruction.

#### BR\_BCLR; BR\_BSET

- 5                   A second set of branch instructions is the branch to a specific label when a specified bit is set or cleared. These instructions set condition codes in the processor status register. The instruction formats are `br_bclr[reg, bit_position, label#]`, `optional_token`; `br_bset[reg, bit_position, label#]`, `optional_token`
- 10                   The operand field Reg A is a context-relative transfer register or general-purpose register that holds the operand. Bit\_position A is a number specifying a bit position in a longword. Bit 0 is the least significant bit. Valid bit\_position values are 0 through 31. Label# is the symbolic label corresponding to the address of a target instruction.
- 15                   This set of branch instructions also has the optional\_tokens; defer one instruction, defer two instructions, and defer three instructions branch guess, as described above.
- Example: `br_bclr[reg, bit_position, label#]`, defer [1] Branch if bit is clear.
- 20   BR=BYTE; BR!=BYTE
- A third set of branch instructions are instructions that cause the processor to branch to the instruction at a specified label if a specified byte in a longword matches or mismatches a byte\_compare\_value. The `br=byte` instruction prefetches the instruction for the "branch taken" condition rather than the next sequential instruction. The `br!=byte`
- 25   instruction prefetches the next sequential instruction. These instructions set the condition codes.
- Format `br=byte[reg, byte_spec, byte_compare_value, label#]`, `optional_token`
- `br!=byte[reg, byte_spec, byte_compare_value, label#]`, `optional_token`
- Reg A is the context-relative transfer register or general-purpose register
- 30   that holds the operand. Byte\_spec Number is a number specifying a byte in register to be compared with byte\_compare\_value. Valid byte\_spec values are 0 through 3. A value of 0 refers to the rightmost byte. Byte\_compare\_value is a parameter used for comparison.

In this implementation, valid `byte_compare_values` are 0 to 255. `Label#` is the symbolic label corresponding to the address of an instruction.

This set of branch instructions also has the optional `_tokens`; defer one instruction, defer two instructions, defer three instructions, and branch guess. Also defer two and three are only used with the `BR!=BYTE`.

Example: `br!=byte[reg, byte_spec, byte_compare_value, label #], defer[3]`

This microword instruction provides a technique for comparing an aligned byte of a register operand to an immediate specified byte value, where `byte_spec` represents the aligned byte to compare (0 is right-most byte, 3 is leftmost byte). The ALU condition codes are set by subtracting the specified byte value from the specified register byte. If the values match, the specified branch is taken. There is a 3 cycle branch latency associated with this instruction therefore, branch deferments of 0, 1, 2 or 3 are allowed in order to fill the latency with useful work. The register can be an A or B bank register.

#### 15 CTX\_ARB

Referring to FIG. 5B, the context swap instruction `CTX_ARB` swaps a currently running context in a specified microengine out to memory to let another context execute in that microengine. The context swap instruction `CTX_ARB` also wakes up the swapped out context when a specified signal is activated. The format for the context swap instruction is:

`ctx_arb[parameter], optional_token`

The “parameter” field can have one of several values. If the parameter is specified as “sram Swap”, the context swap instruction will swap out the current context and wake it up when the thread’s SRAM signal is received. If the parameter is specified as “sram Swap”, the context swap instruction will swap out the current context and wake it up when the thread’s SDRAM signal is received. The parameter can also be specified as “FBI” and swap out the current context and wake it up when the thread’s FBI signal is received. The FBI signal indicates that an FBI CSR, Scratchpad, TFIFO, or RFIFO operation has completed.

The parameter can also be specified as “seq\_num1\_change/seq\_num2\_change”, which swaps out the current context and wakes it up when the value of the sequence number changes. The parameter can be specified as “inter\_thread” which swaps out the current context and wakes it up when the threads

interthread signal is received, or “voluntary” which will swap out the current context if another thread is ready to run, otherwise do not swap. If the thread is swapped, it is automatically re-enabled to run at some subsequent context arbitration point. The parameter can be “auto\_push” which swaps out the current context and wakes it up when  
 5 SRAM transfer read register data has been automatically pushed by the FBus interface, or a “start\_receive” that swaps out the current context and wake it up when new packet data in the receive FIFO is available for this thread to process.

The parameter can also be “kill” which prevents the current context or thread from executing again until the appropriate enable bit for the thread is set in a  
 10 CTX\_ENABLES register, “pci” which swaps out the current context and wake it up when the PCI unit signals that a DMA transfer has been completed.

The context swap instruction CTX\_ARB can have the following optional\_token, defer one which specifies that one instruction will be executed after this reference before the context is swapped.

15 BR=CTX, BR!=CTX

Referring to FIG. 5C, context branch instructions BR=CTX, BR!=CTX are shown. The context branch instruction causes a processor, e.g., microengine 22f to branch to an instruction at a specified label based on whether or not the current executing context is the specified context number. As shown in FIG. 5C, the context branch  
 20 instruction is determined from the branch mask fields when equal to “8” or “9.” The context branch instruction can have the following format:

Format: br=ctx[ctx, label#], optional\_token

br!=ctx[ctx, label#], optional\_token

Label# is a symbolic label corresponding to the address of an instruction.  
 25 Ctx Context number is the number of a context (thread). In this example, valid ctx values are 0, 1, 2, or 3.

This instruction has an optional token “defer one” instruction which causes the processor to execute the instruction following this instruction before performing the branch operation.

30 BR\_INP\_STATE

Referring back to FIG. 5A, another class of branch instructions causes the processor to branch if the state of a specified state name is set to 1. A state is set to 1 or 0 by a microengine in the processor and indicates the currently processing state. It is

available to all microengines. A format as shown in FIG. 5 uses br\_mask field is used to specify branch. For branch mask = 15, the extended field is used to specify the various signal and state signals as listed below.

Format: br\_inp\_state[state\_name, label#], optional\_token

- 5                   Label# is a symbolic label corresponding to the address of an instruction. State\_name is the state name, if rec\_req\_avail when set, it indicates that the RCV\_REQ FIFO has room available for another receive request. If push\_protect, when set, it indicates that the FBI unit is currently writing to the SRAM transfer registers. This instruction can also have an optional\_token "defer one" which executes the instruction
- 10   following this instruction before performing the branch operation.

#### BR\_!SIGNAL

- Another class of branch instruction causes the processor to branch if a specified signal is deasserted. If the signal is asserted, the instruction clears the signal and
- 15   does not take the branch. The SRAM and SDRAM signals are presented to the microengine two cycles after the last Long word is written to the transfer register. The second from last Long word is written 1 cycle after the signal. All other Long words are valid when the signal is submitted. With this instruction, the programmer times the reading of transfer registers appropriately to ensure that the proper data is read.

- 20   Format: br\_signal[signal\_name, label#], optional\_token

Label# is a symbolic label corresponding to the address of an instruction. Signal\_name can be sram, sdram, fbi, pci, inter\_thread, auto\_push start\_receive, seq\_num1, seq\_num2.

- This set of branch instructions has the following optional\_tokens defer one
- 25   instruction, defer two instructions, and defer three instructions. These optional tokens cause the processor to execute one, two, or three instructions following this instruction before performing the branch operation. The ALU operation that sets the condition codes may occur several instructions before the branch instruction. The defer two and defer three are not used with branch guess.

- 30                   A fourth optional token is the guess\_branch Prefetch. The guess branch prefetch token prefetches the instruction for the "branch taken" condition rather than the next sequential instruction. This token is used with defer one instruction to improve performance.

Example: .xfer\_order \$xfer0 \$xfer1 \$xfer2 \$xfer3

sram[read, \$xfer0, op1, 0, 2], sig\_done

wait#:

br\_!signal[sram, wait#], guess\_branch

5 nop;delay 1 cycle before reading \$xfer0

alu[gpr0,0,b,\$xfer0];valid data is written to gpr0

alu[gpr1,0,b,\$xfer1];valid data is written to gpr0

self#:

br[self#].

10 Referring to FIG. 6, the two register address spaces that exist are Locally accessibly registers, and Globally accessible registers accessible by all microengines. The General Purpose Registers (GPRs) are implemented as two separate banks (A bank and B bank) whose addresses are interleaved on a word-by-word basis such that A bank registers have lsb=0, and B bank registers have lsb=1. Each bank is capable of performing a simultaneous read and write to two different words within its bank.

15 Across banks A and B, the register set 76b is also organized into four windows 76b<sub>0</sub>-76b<sub>3</sub> of 32 registers that are relatively addressable per thread. Thus, thread\_0 will find its register 0 at 77a (register 0), the thread\_1 will find its register\_0 at 77b (register 32), thread\_2 will find its register\_0 at 77c (register 64), and thread\_3 at 77d (register 96). Relative addressing is supported so that multiple threads can use the exact same control store and locations but access different windows of register and perform different functions. The use of register window addressing and bank addressing provide the requisite read bandwidth while using only dual ported RAMS in the microengine 22f.

25 These windowed registers do not have to save data from context switch to context switch so that the normal push and pop of a context swap file or stack is eliminated. Context switching here has a 0 cycle overhead for changing from one context to another. Relative register addressing divides the register banks into windows across the address width of the general purpose register set. Relative addressing allows access any of the windows relative to the starting point of the window. Absolute addressing is also supported in this architecture where any one of the absolute registers may be accessed by any of the threads by providing the exact address of the register.

30 Addressing of general purpose registers 78 can occur in 2 modes depending on the microword format. The two modes are absolute and relative. In

absolute mode, addressing of a register address is directly specified in 7-bit source field (a6-a0 or b6-b0), as shown in Table 14:

Table 14

5	7 6 5 4 3 2 1 0
	+---+---+---+---+---+---+---+
	A GPR:   a6  0   a5  a4  a3  a2  a1  a0  a6=0
	B GPR:   b6  1   b5  b4  b3  b2  b1  b0  b6=0
	SRAM/ASB:  a6  a5  a4  0   a3  a2  a1  a0  a6=1, a5=0, a4=0 SDRAM:
10	a6  a5  a4  0   a3  a2  a1  a0  a6=1, a5=0, a4=1

register address directly specified in 8-bit dest field (d7-d0) Table 15:

Table 15

15	7 6 5 4 3 2 1 0
	+---+---+---+---+---+---+---+
	A GPR:   d7  d6  d5  d4  d3  d2  d1  d0  d7=0, d6=0
	B GPR:   d7  d6  d5  d4  d3  d2  d1  d0  d7=0, d6=1
20	SRAM/ASB:  d7  d6  d5  d4  d3  d2  d1  d0  d7=1, d6=0, d5=0
	SDRAM:   d7  d6  d5  d4  d3  d2  d1  d0  d7=1, d6=0, d5=1

If <a6:a5>=1,1, <b6:b5>=1,1, or <d7:d6>=1,1 then the lower bits are interpreted as a context-relative address field (described below). When a non-relative A or B source address is specified in the A, B absolute field, only the lower half of the SRAM/ASB and SDRAM address spaces can be addressed. Effectively, reading absolute SRAM/SDRAM devices has the effective address space; however, since this restriction does not apply to the dest field, writing the SRAM/SDRAM still uses the full address space.

In relative mode, addresses a specified address is offset within context space as defined by a 5-bit source field (a4-a0 or b4-b0)Table 16:

Table 16

	7	6	5	4	3	2	1	0
5	+---+---+---+---+---+---+---+							
	A GPR:   a4  0  context  a3  a2  a1  a0  a4=0							
	B GPR:   b4  1  context  b3  b2  b1  b0  b4=0							
	SRAM/ASB: ab4  0  ab3 context  b2  b1 ab0  ab4=1, ab3=0							
	SDRAM:  ab4  0  ab3 context  b2  b1 ab0  ab4=1, ab3=1							

10

or as defined by the 6-bit dest field (d5-d0) Table 17:

Table 17

	7	6	5	4	3	2	1	0
15	+---+---+---+---+---+---+---+							
	A GPR:   d5  d4 context  d3  d2  d1  d0  d5=0, d4=0							
	B GPR:   d5  d4 context  d3  d2  d1  d0  d5=0, d4=1							
	SRAM/ASB:  d5  d4  d3 context  d2  d1  d0  d5=1, d4=0, d3=0							
20	SDRAM:   d5  d4  d3 context  d2  d1  d0  d5=1, d4=0, d3=1							

25

If <d5:d4>=1,1, then the destination address does not address a valid register, thus, no dest operand is written back.

Other embodiments are within the scope of the following claims.

What is claimed is:

1. A computer instruction comprises:  
a branch instruction that causes a branch in execution of an instruction stream based on any specified value being true or false and including a token that specifies the number of instructions in the instruction stream that are after the instruction  
5 to execute before performing the branch operation.
2. The instruction of claim 2 wherein the branch instruction includes a second token that specifies a branch guess operation.
- 10 3. The instruction of claim 1 wherein the optional token includes defer\_i which causes the processor to execute the i<sup>th</sup> instruction following the branch instruction before performing the branch operation.
4. The instruction of claim 1 wherein the optional token can specify, one, two  
15 or three instructions following branch instruction to execute before performing the branch operation.
5. The instruction of claim 1 wherein the instruction has a format as:  
br [label#], optional\_token,  
20 where br refers to a branch operation, label# is a symbolic representation of an address to branch to and option\_token specifies the number of instructions to execute before performing the branch operation specified by the br branch instruction.
6. The instruction of claim 1 wherein one of the optional tokens are specified  
25 by a programmer or assembler program to enable variable cycle deferred branching.
7. The instruction of claim 1 wherein one of the optional tokens are specified to assist an assembler program to produce more efficient code.
- 30 8. The instruction of claim 1 wherein the branch instruction is a branch unconditionally or branch to an instruction at a specified label based on an ALU condition code.



9. The instruction of claim 1 wherein the instruction is a branch to a specific label when a specified bit is set or cleared.
10. The instruction of claim 1 wherein the branch instruction is a branch instruction that causes the processor to branch to the instruction at a specified label if a specified byte in a longword matches or mismatches a byte\_compare\_value.
11. The instruction of claim 1 wherein the branch instruction is a branch instruction that causes the processor to branch to the instruction at a specified label based on whether or not a current context is a specified context in the branch instruction.
12. The instruction of claim 1 wherein the branch instruction a branch instruction that causes the processor to branch if the state of a specified state name is a selected value.
13. The instruction of claim 1 wherein the branch instruction a branch instruction that causes the processor to branch if a specified signal is deasserted.
14. The instruction of claim 1 wherein the branch instruction further includes an additional token, a guess\_branch token which causes the processor to prefetch the instruction for the "branch taken" condition rather than the next sequential instruction.
15. A computer program comprising:  
a plurality of instructions to cause a processor to execute computer instructions to perform a function, said program having a second plurality of branch instructions in different parts of the program that include different defer branch tokens where the branch tokens specify the number of instructions to execute before performing branch operations specified by the second plurality of branch instructions.
16. The product of claim 15 wherein at least one the branch instructions further includes an additional token, a guess\_branch token which causes the processor to prefetch the instruction for the "branch taken" condition rather than the next sequential instruction.

17. A method of operating a processor comprises:  
executing a branch instruction that causes a branch in an instruction stream  
based on any specified value being true or false; and  
5                   deferring performance of the branch operation of the branch instruction  
based on evaluating a token that specifies the number of instructions to execute before  
performing the branch operation.
18. The method of claim 17 further comprising:  
10                   evaluating a second token that specifies a branch guess operation, which  
causes the processor to prefetch the instruction for the "branch taken" condition rather  
than the next sequential instruction.
19. The method of claim 17 wherein the optional token is selectable by a  
15                   programmer.
20. The method of claim 17 wherein the optional token is specified by a  
programmer or assembler program to enable variable cycle deferred branching.
- 20   21. The method of claim 17 wherein one of the optional token is specified to  
assist an assembler program to produce more efficient code.

ABSTRACT

A processor such as a multithreaded based processor (12) is described. The processor (12) includes a computer instruction (20) that is a branch instruction that causes a branch in execution of an instruction stream based on any specified value being true or false. The instruction also includes a token that specifies the number of instruction of instructions following the branch instruction to execute before performing the branch operation.

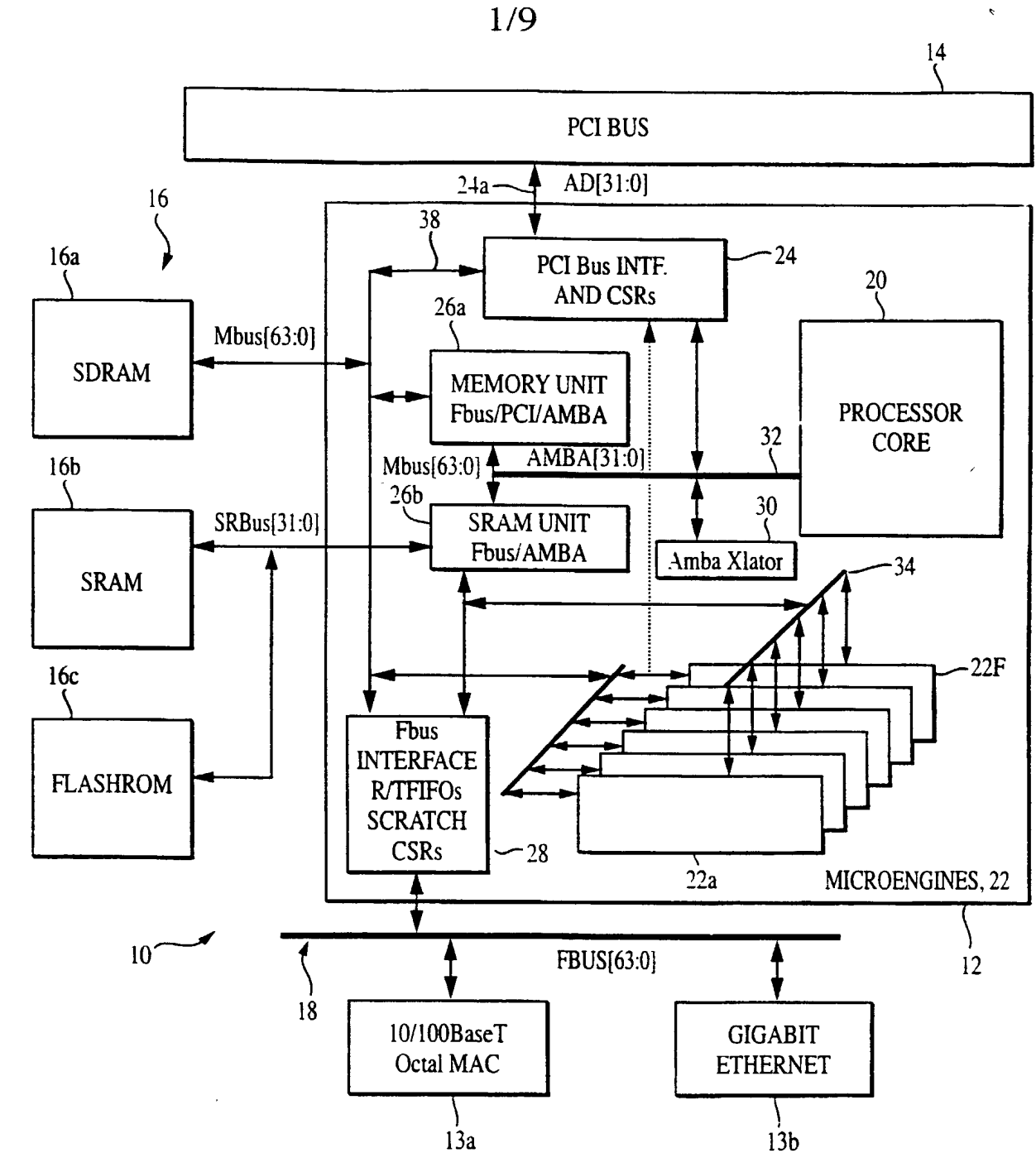
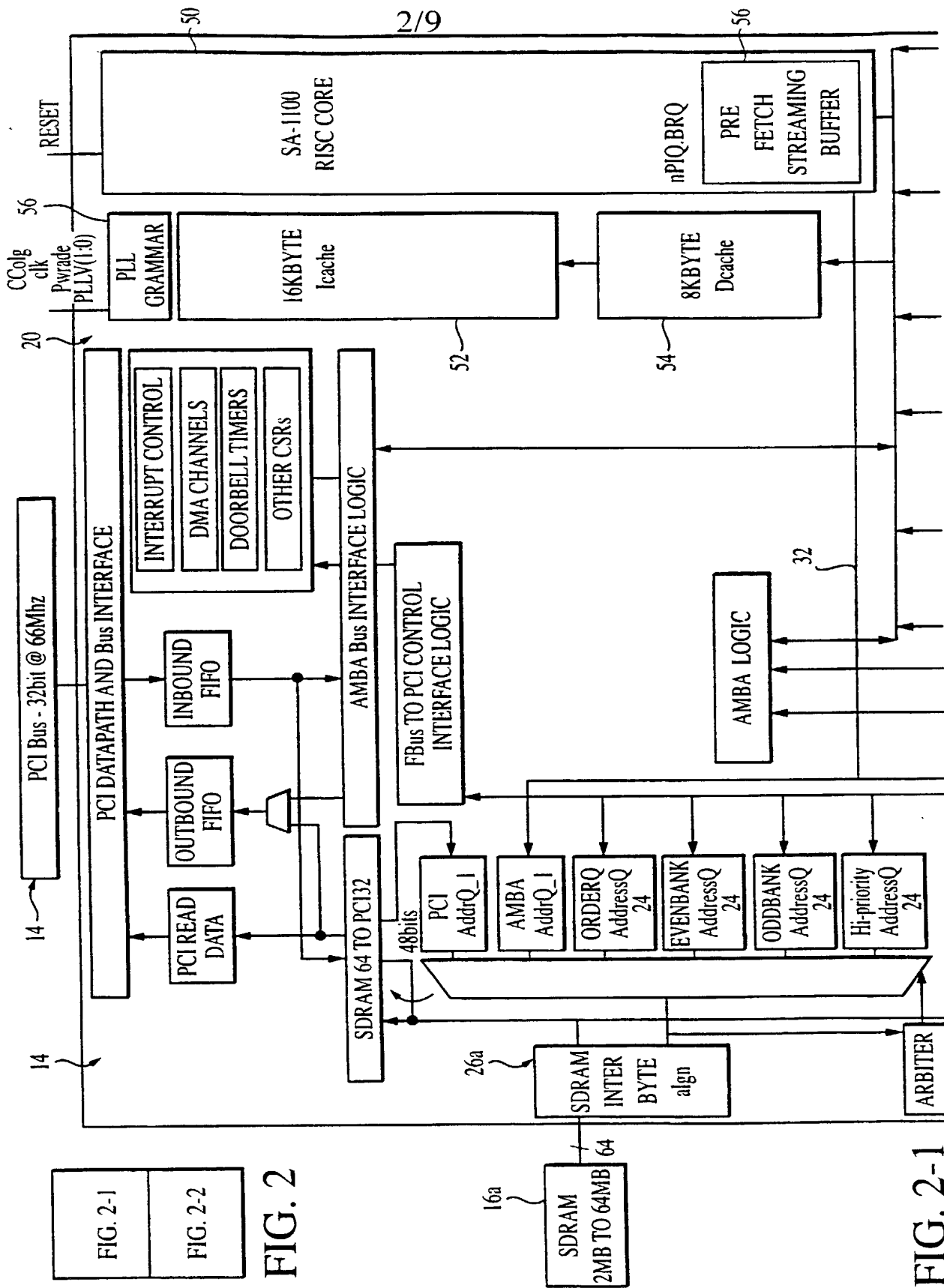
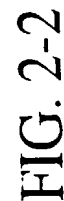


FIG. 1





4/9

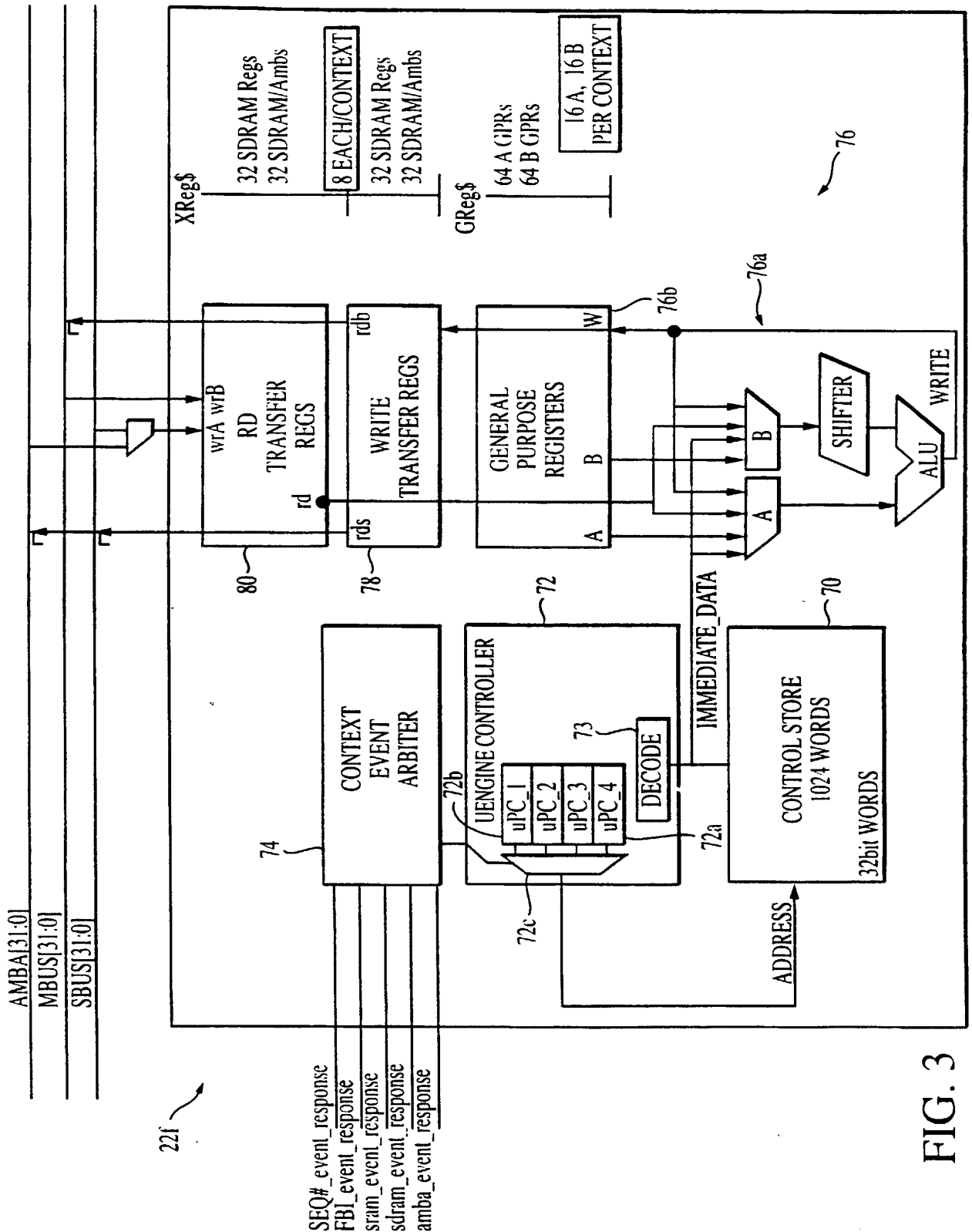


FIG. 3

5/9

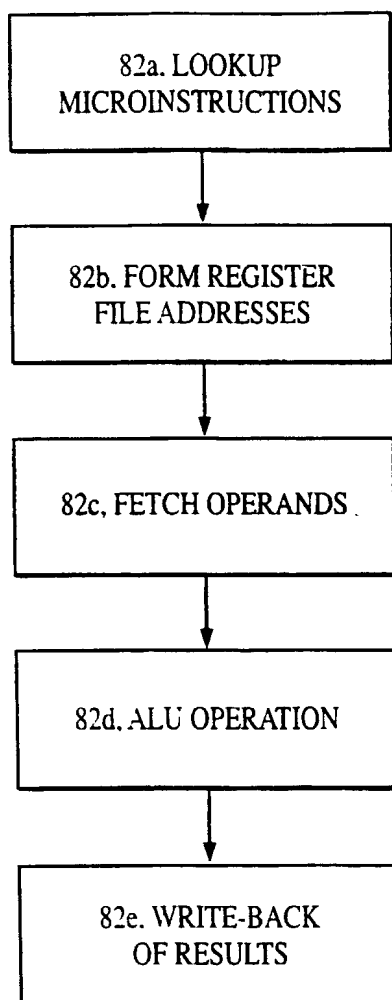
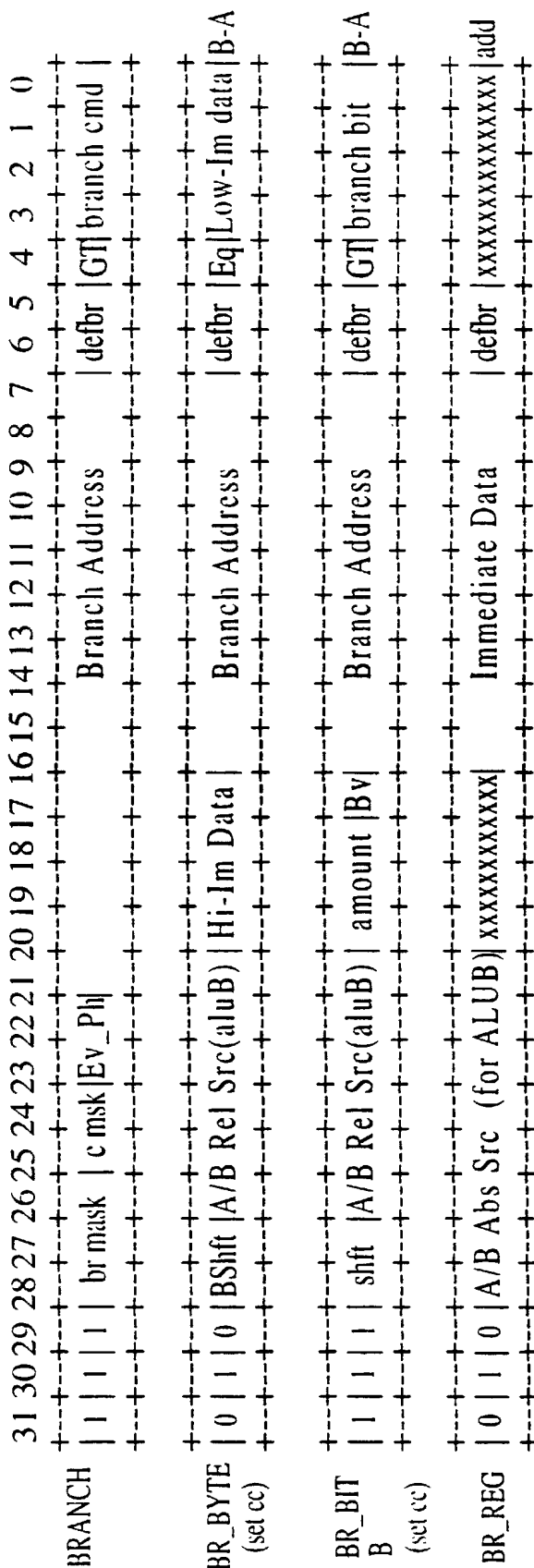


FIG. 4





6/9

# Branch-Mask:

0000 = Zero	0001 = not-zero	1010 = mem-lock
0010 = sign-set	0011 = sign-clear	1100 = unconditional
0100 = carry-out	0111 = not-GT	
0101 = GT	1001 = not-ctx	
1000 = ctx		

## Branch Descriptions:

Defbr => gives number of deferred instructions (must be less/equal max\_allowed (or BR\_ev field))

Deferred instructions can NOT be branch instructions

Br\_Byte => (EQ assumes GT, NEQ assumes NT)

Can have defer = 3 on conditional branches following br-bit or br-byte

FIG. 5A

7/9

128 = SEQ\_NUM\_LSB

3)  $va \rightarrow$  value of sequence number

**FIG. 5B**

FIG. 5C

9/9

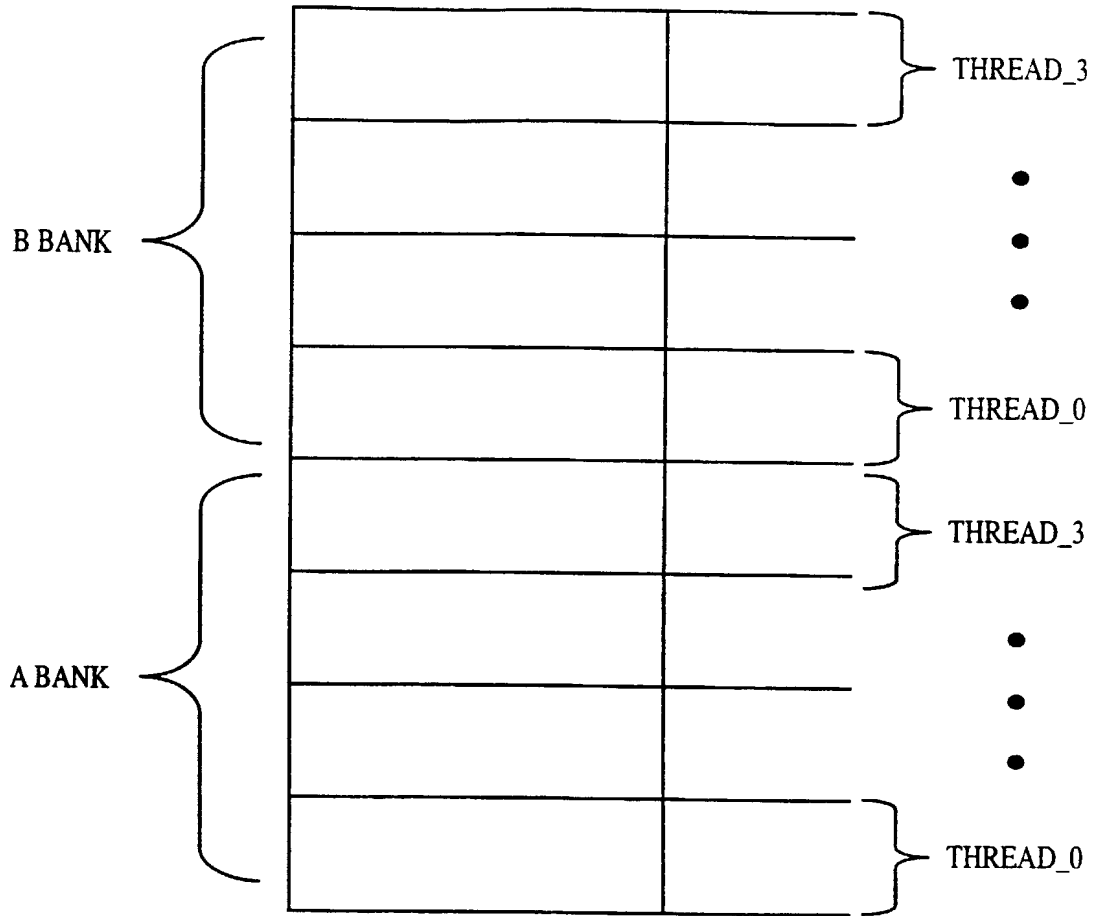


FIG. 6

1005740068 10313502

**POWER OF ATTORNEY  
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)**

**PATENT COOPERATION TREATY  
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE**

The undersigned applicant:

INTEL CORPORATION

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804

☒ as agent(s)

☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR  
Agent's Reference: 10559-311US1  
Application Number: 10/070,008

filed with the United States receiving Office and to make or receive payments on applicant's behalf.

Place: \_\_\_\_\_

Date: \_\_\_\_\_

Signature: \_\_\_\_\_  
INTEL CORPORATION

Printed Name: \_\_\_\_\_

Title: \_\_\_\_\_



**POWER OF ATTORNEY  
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)**

**PATENT COOPERATION TREATY  
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE**

The undersigned applicant:

**GILBERT WOLRICH**

hereby appoints

**Denis G. Maloney, Reg. No. 29,670**

of

**FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804**

☒ as agent(s)

☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: **BRANCH INSTRUCTION FOR PROCESSOR**  
Agent's Reference: **10559-311US1**  
Application Number: **10/070,008**

filed with the United States receiving Office and to make or receive payments on applicant's behalf.

Place:

Hudson, MA US

Date:

June 19, 2002

Signature:

Gilbert Wolrich  
**GILBERT WOLRICH**

Printed Name: GILBERT WOLRICH

Title: Inventor

**POWER OF ATTORNEY  
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)**

**PATENT COOPERATION TREATY  
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE**

The undersigned applicant:

**MATTHEW J. ADILETTA**

hereby appoints

**Denis G. Maloney, Reg. No. 29,670**

of

**FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804**

☒ as agent(s)

☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: **BRANCH INSTRUCTION FOR PROCESSOR**  
Agent's Reference: **10559-311US1**  
Application Number: **10/070,008**

filed with the United States receiving Office and to make or receive payments on applicant's behalf.

Place: Hudson, MA

Date: June 18, 2002

Signature:   
**MATTHEW J. ADILETTA**

Printed Name: MATTHEW J. ADILETTA

Title: Inventor

**POWER OF ATTORNEY  
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)**

**PATENT COOPERATION TREATY  
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE**

The undersigned applicant:

WILLIAM R. WHEELER

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804

☒ as agent(s)

☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with  
the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR  
Agent's Reference: 10559-311US1  
Application Number: 10/070,008

filed with the United States receiving Office and to make or receive payments on applicant's  
behalf.

Place: Hudson, MA USA

Date: 6/19/02

Signature:   
WILLIAM R. WHEELER

Printed Name: WILLIAM R. WHEELER

Title: Inventor



**POWER OF ATTORNEY  
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)**

**PATENT COOPERATION TREATY  
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE**

The undersigned applicant:

DEBRA BERNSTEIN

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804

☒ as agent(s)

☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with  
the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR  
Agent's Reference: 10559-311US1  
Application Number: 10/070,008

filed with the United States receiving Office and to make or receive payments on applicant's  
behalf.

Place: Hudson MA

Date: 6/20/02

Signature: Debra Bernstein  
DEBRA BERNSTEIN

Printed Name: DEBRA BERNSTEIN

Title: Inventor

**POWER OF ATTORNEY  
(CONCERNING A GIVEN INTERNATIONAL APPLICATION)**

**PATENT COOPERATION TREATY  
APPOINTMENT OF AGENT OR COMMON REPRESENTATIVE**

The undersigned applicant:

DONALD F. HOOPER

hereby appoints

Denis G. Maloney, Reg. No. 29,670

of

FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804

☒ as agent(s)

☐ as common representative(s)

to act on applicant's behalf before the competent International Authorities in connection with the international application on

Title: BRANCH INSTRUCTION FOR PROCESSOR

Agent's Reference: 10559-311US1

Application Number: 10/070,008

filed with the United States receiving Office and to make or receive payments on applicant's behalf.

Place: Hudson MA USA

Date: 6/19/02

Signature: Donald F. Hooper  
DONALD F. HOOPER

Printed Name: DONALD F. HOOPER

Title: Inventor

**COMBINED DECLARATION AND POWER OF ATTORNEY**

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name,

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled BRANCH INSTRUCTION FOR PROCESSOR, the specification of which:

☐ is attached hereto.

☒ was filed on February 28, 2002 as Application Serial No. 10/070,008 and was amended on \_\_\_\_\_.

☐ was described and claimed in PCT International Application No. \_\_\_\_\_ filed on \_\_\_\_\_ and as amended under PCT Article 19 on \_\_\_\_\_.

I hereby state that I have reviewed and understand the contents of the above-identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose all information I know to be material to patentability in accordance with Title 37, Code of Federal Regulations, §1.56.

I hereby claim foreign priority benefits under Title 35, United States Code, §119 of any foreign application(s) for patent or inventor's certificate or of any PCT international application(s) designating at least one country other than the United States of America listed below and have also identified below any foreign application for patent or inventor's certificate or any PCT international application(s) designating at least one country other than the United States of America filed by me on the same subject matter having a filing date before that of the application(s) of which priority is claimed:

Country	Application No.	Filing Date	Priority Claimed
WIPO	PCT/US00/24006	08/31/2000	<input checked="" type="checkbox"/> Yes <input type="checkbox"/> No <input type="checkbox"/> Yes <input type="checkbox"/> No

I hereby appoint the following attorneys and/or agents to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith:

Scott C. Harris, Reg. No. 32,030  
David L. Feigenbaum, Reg. No. 30,378  
Christina Sperry, Reg. No. 47,106  
Cathy L. Peterson, Reg. No. 41,249

Denis G. Maloney, Reg. No. 29,670  
Kenneth F. Kozik, Reg. No. 36,572  
Paul A. Pysher, Reg. No. 40,780

Address all telephone calls to DENIS G. MALONEY at telephone number (617) 542-5070.

Address all correspondence to DENIS G. MALONEY at:

FISH & RICHARDSON P.C.  
225 Franklin Street  
Boston, Massachusetts 02110-2804

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patents issued thereon.

**Combined Declaration and Power of Attorney**  
Page 2 of 8 Pages1 - 00 Full Name of Inventor: GILBERT WOLRICHInventor's Signature: *Gilbert Wolrich*Date: June 19, 2002Residence Address: Framingham, MACitizenship: United StatesPost Office Address: 4 Cider Mill Road  
Framingham, MA 017012 - 00 Full Name of Inventor: MATTHEW J. ADILETTAInventor's Signature: *Matthew J. Adiletta*Date: June 19, 2002Residence Address: Worcester, MACitizenship: United StatesPost Office Address: 20 Monticello Drive  
Worcester, MA 016033 - 00 Full Name of Inventor: WILLIAM R. WHEELERInventor's Signature: *William R. Wheeler*Date: 6/19/02Residence Address: Southborough, MACitizenship: United StatesPost Office Address: 9 Darlene Drive  
Southborough, MA 017724 - 00 Full Name of Inventor: DEBRA BERNSTEINInventor's Signature: *Debra Bernstein*Date: 6/20/02Residence Address: Sudbury, MACitizenship: United StatesPost Office Address: 321 Old Lancaster Road  
Sudbury, MA 017765 - 00 Full Name of Inventor: DONALD F. HOOPERInventor's Signature: *Donald F. Hooper*Date: 6/19/02Residence Address: Shrewsbury, MACitizenship: United StatesPost Office Address: 19 Main Circle  
Shrewsbury, MA 01545